1. Для чего используют статические классы?

Статические классы в C# используются для организации кода и предоставления функциональности, которая не требует создания экземпляров класса. Вот основные причины и преимущества использования статических классов:

**1. Группировка связанных методов**

позволяют группировать методы, которые логически связаны между собой, но не требуют состояния объекта.

public static class MathUtils

{

public static int Add(int a, int b)

{

return a + b;

}

}

**2. Отсутствие необходимости в экземплярах**

обеспечивают доступ к методам и свойствам без необходимости создавать объект.

int sum = MathUtils.Add(3, 5);

**3. Общие данные и методы**

могут содержать статические поля и свойства, которые могут хранить общее состояние или данные, доступные для всех пользователей класса.

public static class Configuration

{

public static string DatabaseConnectionString { get; set; }

}

**4. Упрощение кода**

Статические классы могут помочь избежать создания ненужных экземпляров классов, что делает код более простым и понятным

2. Что может содержать статический класс?

 **Статические методы**:

 **Статические свойства**:

 **Статические поля**:

 **Статические конструкторы**: Конструкторы, которые инициализируют статические поля или выполняют другие статические операции при первом обращении к классу

3.Что такое производный и базовый классы?

**Базовый класс** (или родительский класс) — это класс, который предоставляет общие свойства и методы, которые могут быть унаследованы другими классами. Базовый класс служит основой для создания производных классов.

**Производный класс** (или дочерний класс) — это класс, который наследует свойства и методы базового класса. Производные классы могут расширять или изменять функциональность базового класса, добавляя свои собственные свойства и методы.

4. Как используют ключевое слово base?

Ключевое слово base в C# используется для доступа к членам базового класса из производного класса. Оно позволяет обращаться к конструкторам, методам и свойствам базового класса.

5. В чем заключена основная задача наследования?

основная задача наследования заключается в создании иерархий классов, которые способствуют повторному использованию кода, упрощают расширение функциональности, поддерживают полиморфизм

6. Пусть базовый класс содержит метод basefunc(), а производный класс не имеет метода с таким именем. Может ли объект производного класса иметь доступ к методу basefunc()? Если да, то при каких условиях?

Да, объект производного класса может иметь доступ к методу basefunc() базового класса, даже если производный класс не определяет метод с таким именем. Это возможно, если метод basefunc() является **наследуемым** в производном классе и имеет подходящий уровень доступа

(все кроме privat).

7. Напишите объявление конструктора без аргументов для производного класса B, который будет вызывать конструктор без аргументов базового класса A.

public class A

{

// Конструктор без аргументов базового класса

public A()

{

Console.WriteLine("Конструктор A вызван");

}

}

public class B : A

{

// Конструктор без аргументов для производного класса

public B() : base() // Вызов конструктора базового класса

{

Console.WriteLine("Конструктор B вызван");

}

}

8. Что такое полиморфизм? Приведите пример.

Переопределение членов базового класса в классе-наследнике (полиморфизм)

9. Определите назначение виртуальных функций.

Виртуальные функции в C# — это методы, которые могут быть переопределены в производных классах.

### 1. ****Поддержка полиморфизма****

Виртуальные функции позволяют разным классам, которые наследуют от одного базового класса, реализовывать свои собственные версии метода. Это позволяет использовать один и тот же интерфейс для работы с объектами различных типов.

### 2. ****Переопределение поведения****

Когда базовый класс объявляет виртуальный метод, производные классы могут переопределить этот метод, чтобы предоставить специфичное поведение. Это позволяет изменять или расширять функциональность без изменения базового класса.

10. Кому доступны переменные с модификатором protected?

Таким образом, переменные с модификатором protected доступны внутри самого класса и в производных классах

11. Наследуются ли переменные с модификатором private?

нет

12. As, is – что это, как применяется? В чем между ними отличие ?

В C# as и is — это операторы, которые используются для проверки и преобразования типов.

 **Функциональность**:

* is проверяет тип объекта и возвращает true или false.
* as пытается привести объект к указанному типу и возвращает приведенный объект или null.

 **Тип возвращаемого значения**:

* is возвращает логическое значение (bool).
* as возвращает объект, приведенный к указанному типу или null.

 **Обработка ошибок**:

* is безопасен, так как просто проверяет тип, не вызывая исключений.
* as также безопасен, но если приведение невозможно, вместо исключения возвращает null.

13. Поддерживает ли C# множественное наследование?

C# **не поддерживает множественное наследование классов**. Это означает, что класс не может наследоваться от более чем одного базового класса. Однако C# позволяет использовать **множественное наследование интерфейсов**.

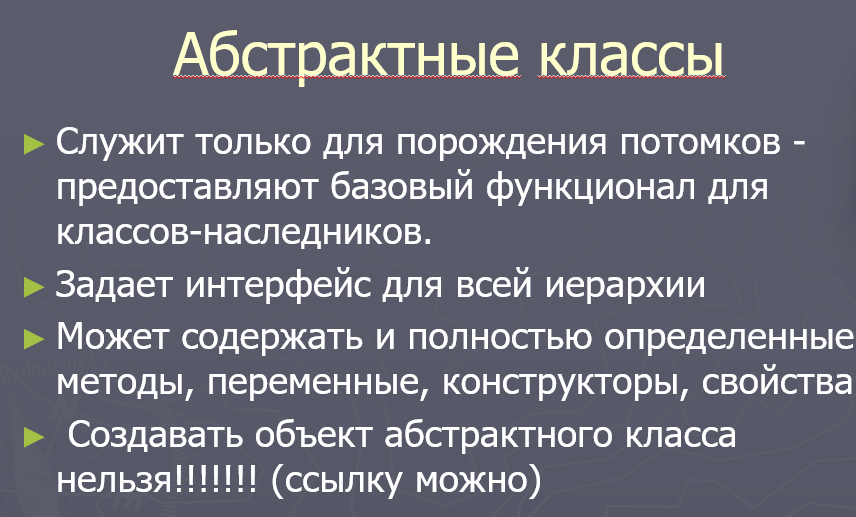
14. Можно ли запретить наследование от класса?

Да, в C# можно запретить наследование от класса, используя модификатор доступа sealed. Когда класс объявлен как sealed, он не может быть унаследован другими классами.

15. Можно ли разрешить наследование класса, но запретить перекрытие метода?

можно разрешить наследование класса, но запретить перекрытие методов, используя модификатор доступа sealed в сочетании с virtual или override.

16. Что такое абстрактный класс?



17. В каком случае вы обязаны объявить класс абстрактным?

Если класс содержит абстрактные методы или свойства

Если класс предназначен только для наследования

18. В чем разница между абстрактными и виртуальными классами? Между виртуальными и абстрактными методами?

### 1. Абстрактные классы vs Виртуальные классы

* **Абстрактные классы**:
  + Не могут быть инстанцированы (т.е. нельзя создать объект).
  + Могут содержать как абстрактные методы (без реализации), так и методы с реализацией.
  + Предназначены для создания базовых классов, от которых будут наследоваться другие классы.
  + Объявляются с помощью ключевого слова abstract.
* **Виртуальные классы**:
  + В C# нет такого понятия, как "виртуальные классы". Виртуальность касается методов, а не классов. Класс может быть обычным или абстрактным, но методы могут быть виртуальными.

### 2. Виртуальные методы vs Абстрактные методы

* **Виртуальные методы**:
  + Объявляются с помощью ключевого слова virtual.
  + Имеют реализацию по умолчанию в базовом классе, но могут быть переопределены в производных классах.
  + Позволяют производным классам использовать базовую реализацию, если это необходимо. }
* **Абстрактные методы**:
  + Объявляются с помощью ключевого слова abstract.
  + Не имеют реализации в базовом классе; производные классы обязаны переопределить их.
  + Используются для определения обязательного интерфейса для производных классов.

19. Какие компоненты класса могут быть виртуальными?

Методы, свойства, индексы

20. Что такое интерфейс?

Интерфейс в C# — это контракт, который Задается набор абстрактных методов, свойств, событий и индексаторов, которые должны быть реализованы в производных классах

Интерфейсы обеспечивают способ определения общего поведения для различных классов без привязки к конкретной реализации.

21. Что может содержать интерфейс?

абстрактные методы, свойства и индексаторы, а также события

Статические поля, методы (с реализацией) и константы (с С# 8.0)

22. Как работать с объектом через унаследованный интерфейс?

 **Определение интерфейса**: Интерфейс IAnimal определяет метод Speak() и свойство Name.

 **Реализация интерфейса**: Классы Dog и Cat реализуют интерфейс IAnimal, предоставляя свои версии метода Speak() и свойства Name.

 **Работа через интерфейс**: В методе Main создаются объекты Dog и Cat, но они объявлены как тип IAnimal. Это позволяет передавать их в метод MakeAnimalSpeak(), который работает с объектами через интерфейс, не заботясь о конкретных реализациях.

IAnimal myDog = new Dog { Name = "Buddy" };

IAnimal myCat = new Cat { Name = "Whiskers" };

23. Приведите пример явной реализации интерфейса.

Явная реализация интерфейса в C# позволяет определять методы интерфейса таким образом, чтобы они могли быть вызваны только через интерфейс, а не через экземпляр класса. Это полезно, когда нужно реализовать несколько интерфейсов, содержащих методы с одинаковыми именами, чтобы избежать конфликтов.

24. Почему нельзя указать модификатор видимости для методов интерфейса?

В C# методы интерфейса не могут иметь модификаторы видимости, потому что все члены интерфейса по умолчанию являются **публичными** и **абстрактными**.

(так ка эти методы должны быть обязательно реализованы)

25. Можно ли наследовать от нескольких интерфейсов?

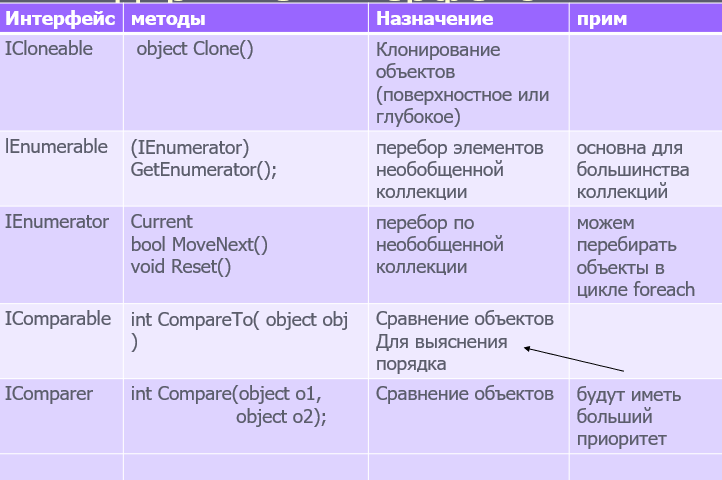
да

26. Назовите отличия между интерфейсом и абстрактным классом.

Интерфейсы и абстрактные классы в C# оба служат для определения контрактов и обеспечения полиморфизма, но у них есть ряд ключевых отличий.



27. Для чего используются стандартные интерфейсы ICloneable, IComparable, IComparer, lEnumerable?



28)2

29)3 4

30)А B

31) 1

32)2

33)А В

34)3

35)4 (Абстрактные классы не могут быть инстанцированы напрямую)

36)4

37)new G